Neural Networks 23 (2010) 135–143

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

An online incremental learning pattern-based reasoning system

Shen Furao [a,*], Akihito Sudo [b], Osamu Hasegawa [b]

[a] *The State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, PR China*

[b] *Imaging Science and Engineering Lab., Tokyo Institute of Technology, Japan*

A R T I C L E   I N F O

Article history:
Received 2 October 2008
Received in revised form 12 May 2009
Accepted 4 June 2009

Keywords:
Pattern-based reasoning
Incremental learning
Real world environment
Clustering

A B S T R A C T

An architecture for reasoning with pattern-based if–then rules is proposed. By processing patterns as real-valued vectors and classifying similar if–then rules into clusters in long-term memory, the proposed system can store pattern-based if–then rules of propositional logic, including conjunctions, disjunctions, and negations. Moreover, it achieves some important properties for intelligent systems such as incremental learning, generalization, avoidance of duplicate results, and robustness to noise. Results of experiments demonstrate that the proposed method is effective for intelligent systems for solving various tasks autonomously in a real environment.

1. Introduction

Reasoning is an important process of human intelligence (Russell & Norvig, 2002). Although many reasoning systems (Lamperti & Zanella, 2006; Meditskos & Bassiliades, 2008; Pan, Yang, & Pan, 2007) have been proposed, they remain insufficient for intelligent systems that must crawl about in the real world, such as robots. One reason is that most systems address only symbol-based if–then rules. Several conventional reasoning systems have been recognized for their effectiveness and have been commercialized, but they perform only in a specific domain. For instance, production systems (Klahr, Langley, & Neches, 1987) are widely sought conventional reasoning systems, but they require human experts to input their knowledge in advance. Consequently, such systems perform well only in a specific domain for which knowledge has been provided by an expert.

In contrast, nobody can correctly predict the complete knowledge list for intelligent systems operating in a varying environment. Systems must therefore learn knowledge that is provided sequentially from environments in an incremental manner. A conventional symbol-based reasoning system must convert learned patterns to symbols before reasoning if it makes inferences using learned knowledge from environments. That is true because it obtains patterns, not symbols, from environments through its sensors. This strategy is apparently impractical now and might be impossible even in the future. Converting patterns to symbols remains a difficult task in spite of numerous studies that have been undertaken for tasks of classification (Bissacco, Chiuso, & Soatto, 2007; Yan, Zhang, Yang, & Hauptmann, 2006). As one example, discriminating cats from dogs is not an easy problem for current discriminators.

Problems will remain even after a classifier with human-like performance is invented. Intelligent systems must generate new symbols when they encounter what should be addressed as a symbol that differs from those they already know. Humans are able to form inferences using patterns obtained from an object. Therefore, intelligent systems must also make inferences using patterns.

Several approaches have been proposed to realize pattern-based reasoning systems. Tsukimoto (2000) proposed a pattern-based reasoning system using perceptrons, which represent learned patterns, as atomic propositions of if–then rules. It can properly address conjunction, disjunction, and negation. However, patterns should be represented with vectors rather than perceptrons for robots because it is difficult to determine when a new perceptron is generated as a new proposition. In Yamane, Hasuo, Suemitsu, and Morita (2007), a pattern-based reasoning system using a non-monotone neural network was proposed. In that system, binary vectors are used as atomic propositions. However, such a system can only properly address implication, not other rules. Moreover, although the system deals with binary vectors, real-valued data must be addressed because sensor information takes a real value. To our knowledge, no report of a study has described a pattern-based reasoning system that employs a real-valued vector as a representation of a pattern, and which can deal with implication, conjunction, disjunction, and negation. In addition, pattern-based reasoning differs completely

* Corresponding address: The State Key Laboratory for Novel Software Technology, Nanjing University, 409-A Mengminwei Bld., 22 Hankou road, 210093 Nanjing, Jiangsu, China. Tel.: +86 25 83686522; fax: +81 45 924 5175.
E-mail address: frshen@nju.edu.cn (S. Furao).

0893-6080/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2009.06.002

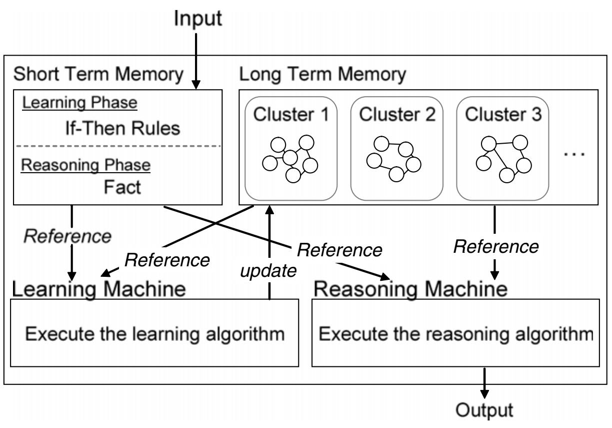

Fig. 1. Architecture of the proposed system.

from fuzzy reasoning because it achieves reasoning without symbols.

As described herein, we propose a pattern-based reasoning system that can process conjunction, disjunction, and negation. The proposed system uses a real-valued vector as a representation of a pattern, i.e., the proposed method can learn if–then rules, and atomic propositions of the rules are patterns represented as real-valued vectors. For instance, the proposed method can learn "((A∧B)∨C)→(D∧¬E∧F)", where A–F are patterns represented as real-valued vectors.[1] After learning if–then rules and obtaining a fact, the proposed system produces an inference from the fact by picking up and connecting learned if–then rules. For example, it outputs "C∨D" and "(E∧¬F)∨D" if the proposed system which has learned "(A∧B)→(C∨D)" and "C→(E∧¬F)" obtains "A∧B" as a fact.

We introduce the proposed method in Section 2. Some experiments to test the efficiency of the proposed method are described in Section 3.

2. Proposed method

As portrayed in Fig. 1, the proposed system consists of short-term memory, long-term memory, a learning machine, and a reasoning machine. The short-term memory stores input data temporarily. The long-term memory stores if–then rules learned from environments. The learning machine executes a learning algorithm when if–then rules are provided. During the learning process, if–then rules are categorized into several clusters in an online manner and unnecessary if–then rules are eliminated. The reasoning machine executes a reasoning algorithm and outputs an OR tree as a reasoning result when a fact is given.

Using the proposed method, in addition to the basic functions for pattern-based reasoning described in Section 1, we can realize some additional functions that are very important for application to an intelligent system such as a robot operating in dynamic environments: incremental learning, generalization ability, and avoidance of duplication of reasoning results.

Incremental learning of if–then rules is necessary because a user cannot provide complete knowledge for a system in advance. The main point of incremental learning is how to learn if–then rules that are provided incrementally without collapsing previously learned knowledge. To the best of our knowledge, no existing method is able to realize incremental learning for pattern-based reasoning. For example, using the method proposed in Yamane et al. (2007), about 20% of reasoning is incorrect when it includes incrementally learned new if–then rules that are the same amount of if–then rules learned previously. The proposed method can realize incremental learning for if–then rules very well. During learning, if an if–then rule is judged as a new rule, the new rule will be added to long-term memory. If the rule is the same as the previously stored learned rule, it will not be added. Then only tuning for long-term memory will occur.

In a real environment, robots rarely obtain data that are identical to previously learned data. The proposed system can find similarly learned if–then rules when different data are given. Moreover, the proposed method has flexibility to let a user configure the degree to which the system will recognize different data as identical data. Other pattern-based reasoning methods have no such flexibility.

The proposed system categorizes learned if–then rules in an online manner to avoid duplication of reasoning results. For some traditional methods, even if the if–then rules are very similar, such similar if–then rules are used as different rules, and an input fact will lead to output of many duplicated reasoning results. For example, given two if–then rules A→B and A′ → B′, and (A, A′) and (B, B′) are similar pattern pairs. For traditional methods, when a fact A is presented, both B and B′ will be output as the reasoning results even they can be interpreted as the same result. Our approach will categorize if–then rules, not individual patterns, to different clusters. During the learning process, similar if–then rules such as A→B and A′ → B′ are categorized to the same cluster. The proposed system will only output B as the reasoning result when a fact A is presented. If we presume that the label of a cluster is the symbol of the patterns within the cluster, then we know that with the categorization function, the proposed method makes intelligent systems to generate symbols as a result of its own reasoning.

2.1. Learning algorithm

The learning machine executes the learning algorithm when the system obtains an if–then rule. Fig. 2 portrays a flowchart of the learning process. The input if–then rule is resolved into several basic if–then rules. Both the conditional part and the sequential part of the basic rule are conjunctive of literals. The resolution is accomplished as follows: both the conditional part and the sequential part are transformed to disjunctive normal forms. Then every combination of clauses is selected as a basic if–then rule. For example, if "(A∧B)∨(C∧¬D)→(¬E∧F)" is input, it is resolved to two basic if–then rules "(A∧B)→(¬E∧F)" and "(C∧¬D)→(¬E∧F)". A conditional part and a sequential part of any if–then rule can be transformed to a disjunctive normal form because any form of a logical formula can be transformed to a disjunctive normal form. The basic if–then rules are stored temporarily in short-term memory.

The system would suffer a high computational load and duplication of reasoning results if all basic if–then rules were simply stored in long-term memory. The following steps of Fig. 2 let the learning machine determine which basic if–then rule should be stored in long-term memory, and which should be conducted. These steps are executed repeatedly for every basic if–then rule. We use a "learning datum" to denote a single basic if–then rule.

First, the most similar if–then rule (first winner) and the second-most similar if–then rule (second winner) to a learning datum are sought among the if–then rules in long-term memory, which are of the same form as the learning datum. Here, the same form means that the number of positive and negative propositions of the conditional part and the sequential part are mutually identical. For example, "A∧¬B→C" is regarded as having the same form of "D∧¬E→F", but is not regarded as the same form of

[1] We respectively use ∧, ∨, ¬, →, to signify conjunction, disjunction, negation, and implication.

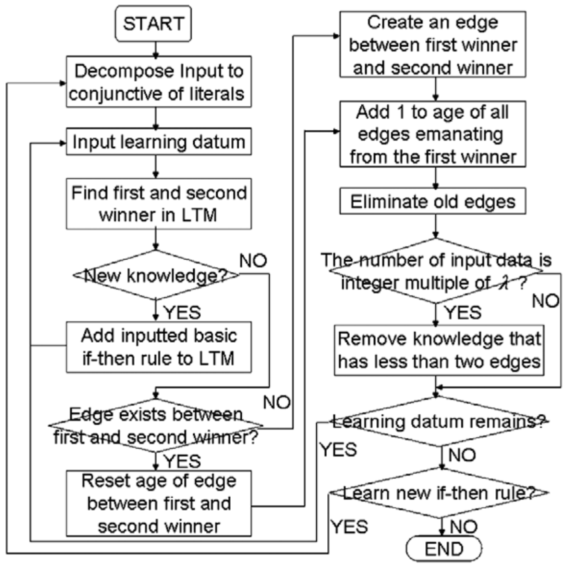
START
Decompose Input to conjunctive of literals
Input learning datum
Find first and second winner in LTM
New knowledge?
NO
YES
Add inputted basic if-then rule to LTM
Edge exists between first and second winner?
NO
YES
Reset age of edge between first and second winner
Create an edge between first winner and second winner
Add 1 to age of all edges emanating from the first winner
Eliminate old edges
The number of input data is integer multiple of λ ?
YES
NO
Remove knowledge that has less than two edges
Learning datum remains?
YES
NO
Learn new if-then rule?
YES
NO
END

Fig. 2. Flowchart of the learning phase.

"¬D∧¬E→F". The similarity between the same-form if–then rules is calculated using the following distance measure d.

$$d(k_i, k_m) = \frac{1}{\sqrt{D}} \left\{ \min_{\sigma_1 \in S_M} \sum_{i=1}^{M} \|P_i - P'_{\sigma_1(i)}\| + \min_{\sigma_2 \in S_N} \sum_{i=1}^{N} \|Q_i - Q'_{\sigma_2(i)}\| + \min_{\sigma_3 \in S_{\tilde{M}}} \sum_{i=1}^{\tilde{M}} \|\tilde{P}_i - \tilde{P}'_{\sigma_3(i)}\| + \min_{\sigma_4 \in S_{\tilde{N}}} \sum_{i=1}^{\tilde{N}} \|\tilde{Q}_i - \tilde{Q}'_{\sigma_4(i)}\| \right\}. \quad (1)$$

Therein, $S_M, S_N, S_{\tilde{M}}$, and $S_{\tilde{N}}$ respectively signify the symmetric group of degree M, N, $\tilde{M}$, and $\tilde{N}$, and D represents the dimension of the patterns; the learning datum k and an if–then rule k' stored in long-term memory are denoted respectively as follows.

$$\left(\bigwedge_{i=1}^{M} P_i\right) \wedge \left(\bigwedge_{j=1}^{N} \neg Q_j\right) \to \left(\bigwedge_{i=1}^{\tilde{M}} \tilde{P}_i\right) \wedge \left(\bigwedge_{j=1}^{\tilde{N}} \neg \tilde{Q}_j\right) \quad (2)$$

$$\left(\bigwedge_{i=1}^{M} P'_i\right) \wedge \left(\bigwedge_{j=1}^{N} \neg Q'_j\right) \to \left(\bigwedge_{i=1}^{\tilde{M}} \tilde{P}'_i\right) \wedge \left(\bigwedge_{j=1}^{\tilde{N}} \neg \tilde{Q}'_j\right). \quad (3)$$

We determine whether a learning datum is a member of the same cluster of both the first winner and the second winner or not. The learning datum is regarded as a member of the same cluster of a stored if–then rule if the distance between the rule and the learning datum is less than the similarity threshold of the rule. The similarity threshold of an if–then rule k_w is calculated as follows.

$$s = \begin{cases} \max_{k \in N} d(k, k_w) & (\text{if } N \neq \emptyset) \\ \min_{k \in A} d(k, k_w) & (\text{if } N = \emptyset). \end{cases} \quad (4)$$

Therein, N is the set of all the if–then rules connected to the k_w rule by an edge, A is the set of all the if–then rules in long-term memory.

The learning datum is added to long-term memory as new knowledge. The learning process for the learning datum finishes if the learning datum is not a member of the same cluster of the first winner or the second winner. If the learning datum is a member of the same cluster of the first winner or the second winner, the learning datum is not added to long-term memory and the previously stored knowledge in long-term memory is updated as follows: An edge between the first winner and the second winner is generated if the edge has not been generated yet; the age of the edge is set to zero. Subsequently, the age of every edge emanating from the first winner is increased by 1. Every edge with age greater than a parameter Λ_{edge} is removed.

Here, the edge is used to generate clusters of if–then rules in long-term memory. If–then rules connected with an edge are regarded as members of the same cluster. In online learning, the basic if–then rules belonging to the same cluster at an early stage might not belong to same cluster at a more advanced stage. It therefore becomes necessary to remove edges that have not been refreshed recently. We use the 'age' of the edge to judge whether the edge between two if–then rules will be removed.

We can define the within-cluster distance d_w of cluster C and the between-cluster distance $d_b(C_i, C_j)$ of clusters C_i and C_j as

$$d_w = \frac{1}{N_C} \sum_{(i,j) \in C} d(i, j), \quad (5)$$

$$d_b(C_i, C_j) = \min_{i \in C_i, j \in C_j} d(i, j). \quad (6)$$

Here, N_C is the number of rules in cluster C. From the definition of similarity threshold s (formula (4)), the similarity threshold s of k_w is greater than the within-cluster distance of the cluster to which k_w belongs. In addition, s is less than the between-cluster distance of k_w to other clusters. Therefore, at any time, when new data k is input to the system, the similarity threshold s of winner k_w ensures that k is processed correctly. It is then inserted to long-term memory as new knowledge or is used to tune long-term memory: the automatically updated similarity threshold eliminates the influence of the order in the presentation of new data. Therefore, incremental learning can be realized well.

If–then rules in long-term memory which have less than two edges are removed if the number of the previously provided learning data is an integer multiple of a parameter λ. This step is expected to eliminate noise from the learning data. Here, noise signifies learning data whose features comprise un-useful noisy elements. We assume that noise lies in a low-density area. If an if–then rule has less than two edges (i.e., the if–then rule has few neighbors), it therefore lies in a low-density area. We remove it because it might be generated by noise.

Algorithm 1. Learning phase of the proposed method

```
 1: Input an if–then rule $r$.
 2: Decompose the input if–then rule $r$ to basic if–then rules
    $a_1, a_2, \ldots, a_n$.
 3: for $l = 1$ to $L$ do
 4:   Randomly choose a basic if–then rule $a_j$ from $a_1, a_2, \ldots, a_n$.
 5:   Find first winner $k_1$ and second winner $k_2$ in long-term
      memory $A$ by
      $k_1 = \arg\min_{i \in A} d(a_j, k_i)$ and $k_2 = \arg\min_{i \in A \setminus k_1} d(a_j, k_i)$.
 6:   Calculate threshold $s_{k_1}$ of $k_1$ and $s_{k_2}$ of $k_2$ using formula (4).
 7:   if $d(a_j, k_1) > s_{k_1}$ and $d(a_j, k_2) > s_{k_2}$ then
 8:     Add $a_j$ as a new if–then rule to long-term memory $A$.
 9:   else
10:     if An edge $(k_1, k_2)$ exists between $k_1$ and $k_2$ then
11:       $age_{(k_1,k_2)} = 0$
12:     else
13:       Create an edge $(k_1, k_2)$ between $k_1$ and $k_2$.
14:     end if
15:     Increase the age of all edges linked with $k_1$ by 1.
16:     Find the edges whose ages are greater than $\Lambda_{edge}$, then
        remove such edges.
17:     if $l$ is an integer multiple of λ then
18:       Remove if–then rules in $A$ with fewer than two edges.
19:     end if
```

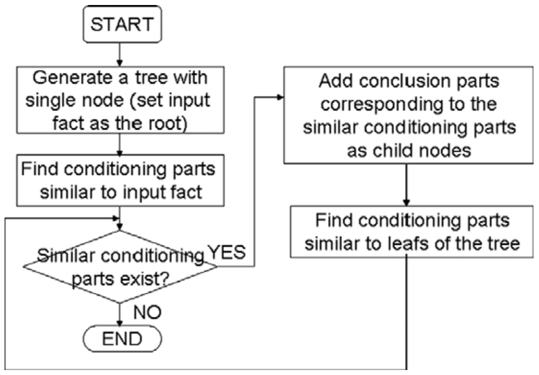
START
Generate a tree with single node (set input fact as the root)
Find conditioning parts similar to input fact
Similar conditioning parts exist?
YES
NO
END
Add conclusion parts corresponding to the similar conditioning parts as child nodes
Find conditioning parts similar to leafs of the tree

Fig. 3. Reasoning phase flowchart.

```
20:      end if
21:   end for
22:   Return generated long-term memory $A$ as the learning result.
```

In summary, Algorithm 1 gives details of the learning process. In Algorithm 1, we must determine two parameters Λ_{edge} and λ. These two parameters will influence the frequency of deleting edges between nodes of if–then rule and rules lie in sparse area. Consequently, we choose a large value for these two parameters and obtain many if–then rule nodes inside one cluster to realize reasoning with high precision if we want to save previous learned knowledge much longer. In contrast, we set the value of these two parameters small to remove if–then nodes and edges frequently if we desire fewer nodes to save memory space and speed up reasoning. Therefore, the two parameters depend on the real condition of the task. We can use these parameters to control the performance of the proposed method.

2.2. Reasoning algorithm

Fig. 3 depicts a flowchart of the reasoning phase. When a fact f such as A∧B is input, the system generates a tree. The root of the tree is the input fact f. The distance between the fact f and the conditional part c of each if–then rule in long-term memory is calculated using the following distance measure d_r:

$$d_r(f,c) = \frac{1}{\sqrt{D}}\left\{\min_{\sigma_1\in S_M}\sum_{i=1}^{M}\|P_i - P'_{\sigma_1(i)}\| + \min_{\sigma_2\in S_N}\sum_{i=1}^{N}\|Q_i - Q'_{\sigma_2(i)}\|\right\}, \qquad (7)$$

where the fact f and the conditional part c are denoted as

$$\left(\bigwedge_{i=1}^{M} P_i\right) \wedge \left(\bigwedge_{j=1}^{N} \neg Q_j\right), \qquad (8)$$

$$\left(\bigwedge_{i=1}^{M} P'_i\right) \wedge \left(\bigwedge_{j=1}^{N} \neg Q'_j\right). \qquad (9)$$

The sequential (conclusion) part s of the if–then rule is added to the tree as a child of the root if distance $d_r(f,c)$ is less than a predefined parameter δ_r. Only the nearest sequential part is added from the same cluster if several sufficiently close sequential parts exist in the same cluster. Then the above steps are repeated for the new generated child until no new child can be generated. For example, we presume that three if–then rules stored exist in long-term memory: "A∧B→C∨(D∧¬E)", "C′ →F∨G∨(H∧I)", and "D′ ∧ ¬E′ → ¬J", where (A, A′)–(E, E′) are pairs of sufficiently similar patterns. If A′ ∧ B′ is presented as a fact, the reasoning machine will output a tree resembling that shown in Fig. 4.

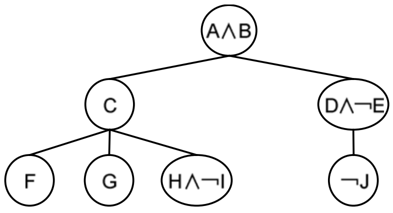

Fig. 4. Example of an OR tree as a result of reasoning.

An output tree should be addressed as an OR tree, and a deduced proposition is a disjunctive normal form generated by combining several propositions in nodes. If the reasoning machine outputs the tree portrayed in Fig. 4, "C∨(D∧¬E)", "C∨¬J", "F∨G∨(H∧I) ∨(D∧¬E)", and "F∨G∨(H∧I) ∨¬J " are correct reasoning results. A reasoning result that is represented as a disjunctive normal form can be interpreted as the possibility of an environment. The system discovers the possibility of places that the system can reach from the present location if the present location is input to the system as a fact and "the supermarket or the station or the drug store" is the reasoning result.

Algorithm 2. Reasoning phase of the proposed method

```
1:  Presume that $n$ clusters exist in long-term memory $A$, and that
    $N$ members exist in $A$: $a_1, a_2, \ldots, a_N$.
2:  Input a fact $f$.
3:  Generate a tree with $f$ as the root.
4:  for $i = 1$ to $N$ do
5:     Calculate $d_r(f, c_i)$, where $c_i$ is the conditional part of $a_i$.
6:     if $d_r(f, c_i) < \delta_r$ then
7:        Add $a_i$ to temporal set $B_k$, all members in $B_k$ have the same
          cluster label with $a_i$.
8:     end if
9:  end for
10: for $k = 1$ to $n$ do
11:    Find the nearest member $b$ of $B_k$ to input fact $f$ by
       $b = \arg\min_{b_j\in B_k} d_r(f, b_j)$.
12:    Add the sequential part $s$ of $b$ to the tree as the child of $f$.
13: end for
14: For all leaves of the tree, repeat the same steps from step4 to
    step13 to add children for leaves until no child can be added to
    the tree.
15: Return the generated OR-tree as the reasoning result.
```

Algorithm 2 gives details of the reasoning process. In Algorithm 2, we must determine one parameter δ_r. This parameter is used to configure the degree to which the system will recognize different data as identical data. Small δ_r means only very similar knowledge in long-term memory can be recognized as the same data from the input fact; very small δ_r causes the loss of generalization of the reasoning system. Large δ_r allows more knowledge to be identified from the input fact, but a much larger value leads to high-error reasoning results.

2.3. Neural network model of the proposed method

The architecture of the proposed system, as shown in Fig. 1, is presented in terms of knowledge engineering; it can also be interpreted as a neural network resembling that shown in Fig. 5. The neural network model of the proposed method has an input layer and a competitive layer. The input layer obtains an if–then rule or a fact. New nodes are generated adaptively when the input layer obtains an if–then rule. Each if–then rule is represented with several nodes connected by edges. The proposed model requires

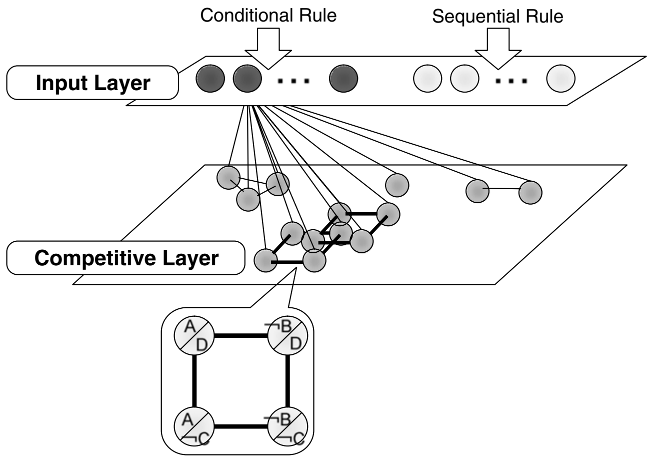

Fig. 5. Neural networks model of the proposed method.

$m \times n$ neurons to represent an if–then rule if the conditional part of an if–then rule has m literals and the sequential part has n literals. Each neuron holds a pair of literals picked up from the conditional part and the sequential part one-by-one, which means that each neuron holds two vectors and flags of negation. Neurons representing $A \wedge \neg B \rightarrow \neg C \wedge D$ are portrayed in Fig. 5.

As described above, the proposed method can realize incremental learning. Incremental learning addresses the ability of repeatedly training a network using new data without destroying the old prototype patterns. Some traditional artificial neural networks such as back-propagation (BP) networks (Haykin, 1998) and radial basis function (RBF) networks (Broomhead & Lowe, 1988) have a fixed number of nodes that the designer must determine. All learned weights must be re-trained when new knowledge is input to the system. In fact, ART networks (Carpenter & Grossberg, 1988) introduce a similarity criterion (vigilance) and allow learning only if the pattern sufficiently matches stored prototypes. The method to determine the vigilance parameter remains difficult. A bad choice can result in a catastrophic allocation of new nodes. Growing neural gas (GNG) (Fritzke, 1995) inserts new nodes to the network using a local-error criterion, this method suffers from a permanent increase in the number of nodes. Lim and Harrison (1997) propose a hybrid network combining advantages of Fuzzy ARTMAP and probabilistic neural networks for incremental learning. Hamker (2001) proposes a life-long learning cell structure (LLCS) that can learn the number of nodes needed to solve a task and to adapt the learning rate of each node separately and dynamically. Both methods work for supervised learning tasks, but the method of processing unsupervised learning and doing clustering remains controversial. A self-organizing incremental neural network (SOINN) (Shen & Hasegawa, 2006) can realize the unsupervised incremental learning task. However, SOINN adopts a two-layer structure, it is difficult to choose when to halt first-layer learning and begin second-layer learning. For the second layer, if the learning results of the first layer were changed, all learned results of the second layer would be destroyed, thereby necessitating re-training of the second layer.

As described herein, using a similarity threshold, the proposed method can insert new nodes to the network to represent new knowledge without destroying learned knowledge. The similarity threshold is determined automatically using the learned knowledge; then the value is adapted to the present situation. During the learning process, when new input patterns are input, the distance between the new pattern and the winner is calculated and compared with the similarity threshold of the winner. Consequently, the knowledge of the new pattern is learned and tuning for the weight of the winner will take place if the distance

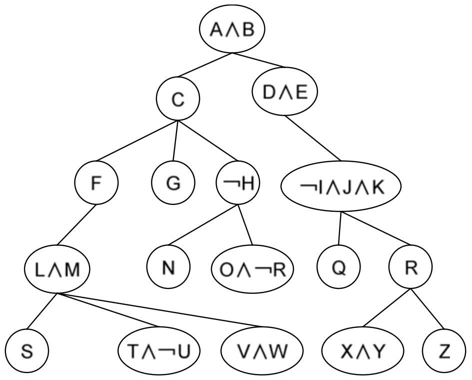

Fig. 6. Desirable output of experiment with artificial data.

is less than the threshold, meaning that the knowledge of the new pattern is not learned if the distance is greater than the similarity threshold. Therefore, a node representing the new pattern will be added to the system. Comparing the methods described above, the proposed method obviates predetermination of the important similarity parameter and obviates retraining of the network when new information comes. Moreover, adaptively updating the similarity parameter assures that the proposed method avoids permanent increases in the number of nodes: the proposed method is well suited to unsupervised incremental learning.

3. Experiment

3.1. Simulation with artificial data

In this experiment, the proposed system learns if–then rules in a sequential manner (not in a batch manner), where atomic propositions of the if–then rules are patterns generated from a Gaussian distribution. Such rules are presented incrementally to the system. The system then produces inferences using the learned if–then rules.

The if–then rules for the system to learn are $A \wedge B \rightarrow C \vee (D \wedge E)$, $C \rightarrow F \vee G \vee \neg H$, $D \wedge E \rightarrow \neg I \wedge J \wedge K$, $F \rightarrow L \wedge M$, $\neg H \rightarrow N \vee (O \wedge \neg R)$, $\neg I \wedge J \wedge K \rightarrow Q \vee R$, $L \wedge M \rightarrow S \vee (T \wedge \neg U) \vee (V \wedge W)$, and $R \rightarrow (X \wedge Y) \vee Z$. The reasoning result can be represented as an OR-tree resembling that shown in Fig. 6 if the reasoning system learns these if–then rules correctly and obtains $A \wedge B$ as a fact.

After resolving the original if–then rules to 16 basic if–then rules, the system learns each "conjunction of literal → conjunction of literal"-formed if–then rule 1000 times. In all cases, A–Z are represented using 200 dimensional vectors; such vectors are generated from a Gaussian distribution, the mean of which is a random value from -1 to 1, and the variance of which is 0.1. Consequently, in all, $16 \times 1000 = 16{,}000$ basic if–then rules exist. Each rule is represented as a pattern (not a symbol). In this experiment, we use different Λ_{edge} and λ to test the performance of the proposed method. For large Λ_{edge} and λ, we obtain numerous nodes for clusters. For small Λ_{edge} and λ, we obtain a small number of nodes for clusters. For all value of Λ_{edge} and λ, the number of clusters is the same; the reasoning results are nearly the same. Here, as an example, we only report the learning results of parameters $\Lambda_{edge} = 100$, $\lambda = 50$.

3.1.1. Result of the learning phase

During the learning phase, the 16,000 if–then rules (represented with real-valued vectors) are input to the system sequentially. New rules are input incrementally to the system to test the

Table 1
Experimental environments for incremental learning.

If–then rule	Environment							
	I	II	III	IV	V	VI	VII	VIII
R_1	1	0	0	0	1	0	0	0
R_2	0	1	0	0	0	1	0	0
R_3	0	1	0	0	0	0	0	0
R_4	0	0	1	0	1	0	0	0
R_5	0	0	1	0	0	0	0	0
R_6	0	0	0	1	0	0	1	0
R_7	0	0	0	1	0	0	0	0
R_8	0	0	0	0	1	0	1	0
R_9	0	0	0	0	1	0	0	0
R_{10}	0	0	0	0	0	1	0	0
R_{11}	0	0	1	0	0	1	0	0
R_{12}	0	0	0	0	0	0	1	0
R_{13}	0	0	0	0	1	0	1	0
R_{14}	0	0	0	1	0	0	1	0
R_{15}	0	0	0	0	0	0	0	1
R_{16}	0	0	0	0	1	0	0	1

Table 2
Learning results obtained using artificial data.

Cluster no.	If–then rule	Number of members	Compression ratio (%)
R_1	A∧B→C	102	10.2
R_2	A∧B→D∧E	98	9.8
R_3	C→F	101	10.1
R_4	C→G	101	10.1
R_5	C→ ¬H	95	9.5
R_6	D∧E → ¬I∧J∧K	96	9.6
R_7	F→L∧M	103	10.3
R_8	¬H→N	98	9.8
R_9	¬H→O∧¬R	99	9.9
R_{10}	¬I∧J∧K→Q	100	10
R_{11}	¬I∧J∧K→R	104	10.4
R_{12}	L∧M→S	91	9.1
R_{13}	L∧M →T∧¬U	99	9.9
R_{14}	L∧M→V∧W	103	10.3
R_{15}	R→X∧Y	96	9.6
R_{16}	R→Z	101	10.1
Total	–	1587	9.9

proposed method. Table 1 presents the incremental learning environment. For steps 1–1000 (Environment I), patterns representing first rule R_1 are input randomly to the system. At step 1001, the environment changes and patterns of new rules R_2 and R_3 are input randomly to the system (Environment II). At step 3001, the environment changes again, etc. In Table 1, for every environment, if–then rules to be used for training are marked as '1'; other rules are marked as '0'.

Table 2 presents learning results. The input rules are categorized into 16 different clusters (Column II of Table 2) in long-term memory. The labels of such clusters are the same rules as those used in a symbol-based manner. The number of members of each cluster is shown in Column III of Table 2, and the number of members representing the 16 basic if–then rules is 1587.

During the experiment, in Environment I the if–then rule R_1 is learned: A∧B→C. The system generates 102 nodes to represent the R_1 cluster. In Environment II, new if–then rules R_2 and R_3 are learned. The system generates 98 nodes for R_2 and 101 nodes for R_3. The remaining nodes of rule R_1 are not destroyed. Such nodes play a major role in subsequent learning. For example, in Environment V, patterns of rule R_1 are input to the system again. No insertion of nodes occurs for cluster R_1 because all knowledge of rule R_1 was learned during learning of Environment I.

Each cluster has members corresponding only to the basic if–then rule of this cluster: A∧B → C∨(D∧E), C→ F∨G∨¬H, D∧E → ¬I∧J∧K, F→L∧M, ¬H→N∨(O∧¬R), ¬I∧J∧K→Q∨R, L∧M→ S∨(T∧¬U)∨(V∧W), and R→ (X∧Y)∨Z. The members of clusters are represented with real-valued patterns. Consequently, the system can categorize if–then rules appropriately into clusters. They are the same rules as those used in a symbol-based manner.

Column IV of Table 2 lists the compression ratio, calculated as N_m/N_l, where N_m is the number of members of the cluster and N_l is the number of if–then rules provided to the system as learning data.

Although the learning data comprise 16,000 if–then rules and although the atomic propositions (real-valued patterns) of such rules mutually differ, only 1587 rules are stored in long-term memory using the proposed method. The system eliminated 14,413 rules because the rules resembled previously stored rules. The total compression ratio was 9.9%.

To test the degree to which the order of presentation of new data influences the learning performance, we also test other orders that differ from those presented in Table 1, such as learning R_4, R_8, and R_{10} at first, then learning R_2, R_5, etc. Using the same parameters $\Lambda_{edge} = 100$ and $\lambda = 50$, the learning results of different orders are nearly identical to the results presented in Table 2. Only a slight change in the number of members for clusters occurs. All rules are categorized appropriately, which shows that the order in the presentation of new data has little influence on the learning results.

The learning results reflect that the proposed method can learn pattern-based if–then rules, categorize such rules to different clusters, learn new rules without collapsing previously learned rules, and avoid memory consumption by eliminating similar patterns.

3.1.2. *Result of reasoning phase*

We provide "A∧B", "C", "D∧E", "F", "G", "¬H", "¬I∧J∧K", "L∧M", "N" "O∧¬R", "Q", "R", "S", "T∧¬U" "V∧W", "X∧Y", and "Z" as facts to test whether the system can produce inferences correctly. Such facts are generated from the same Gaussian distribution as that used in the learning phase. For each fact, 200 patterns are generated from the Gaussian distribution. In all, 3400 facts are provided to the system. Such test patterns differ from the training patterns in the learning phase, but they are generated from the same distribution of the training patterns to test the generalization performance of the proposed method. The reasoning phase parameter is set to $\delta_r = 0.3$.

The reasoning phase results are presented in Table 3. Column I of Table 3 is the fact provided to the system. There are 17 facts and 200 real-valued patterns for every fact used to test the proposed method. The reasoning phase will output an OR-tree when the pattern of a fact is presented. Through comparison of the tree with Fig. 6, we can judge whether this fact is correctly reasoned. Column II of Table 3 lists the number of correct reasons. Column III of Table 3 shows the correct reasoning ratio that is calculated with N_c/N_T, where N_c is the number of correct reasons and N_T is the number of total test patterns. The reasoning ratio is evidence that the proposed system is able to produce an inference well, with an approximately 99.999% correct reasoning ratio. It also proves that the learning phase can categorize input if–then patterns to if–then rule clusters correctly. In fact, the members of such clusters are able to represent the if–then rule well. The correct reasoning ratio remains very high, even if plenty of similar patterns are eliminated.

Results of the experiment show that the system can avoid duplication of results. For example, when a pattern corresponding to A∧B is provided as a fact, then 895,161 nodes are created in the output tree when we force the system to regard every if–then rule in long-term memory as belonging to different clusters. If–then rules are not categorized to different clusters in long-term memory.

From the reasoning results, we know that incremental learning is realized perfectly. The correct reasoning ratio of previously learned if–then rules and later learned if–then rules are nearly equal, which means that the learning of new knowledge does not destroy the learned knowledge.

Table 3
Reasoning results for artificial data.

Provided fact	Number of correct reasons	Correct reasoning ratio (%)
A∧B	200	100
C	200	100
D∧E	199	99.5
F	200	100
G	200	100
¬H	200	100
¬I∧J∧K	199	99.5
L∧M	200	100
N	200	100
O∧¬R	200	100
Q	200	100
R	200	100
S	200	100
T∧¬U	198	99
V∧W	200	100
X∧Y	200	100
Z	200	100
Total	3396	99.99882

3.1.3. Noise robustness and generalization

Using the learned if–then rules presented in Section 3.1.1, we test the proposed method's noise robustness. We provide 10 white-noise samples to examine whether the system can reason correctly from this type of noisy data. Here, the white-noise sample means a zero mean random vector; its autocorrelation matrix is a multiple of the identity matrix.

$$\mu_W = E\{W\} = 0 \tag{10}$$

$$R_{WW} = E\{WW^{\mathrm{T}}\} = \sigma^2 I. \tag{11}$$

With white-noise samples as the fact, we use Algorithm 2 to perform the reasoning process. The system makes no child node of the white noise; it judges that all the white noise consists of unknown patterns. Some other pattern-based system such as Yamane et al. (2007) cannot judge whether an input fact has been learned. It outputs a wrong pattern that is entirely unrelated to learned patterns when it obtains an unknown pattern.

Moreover, we generate noisy data by adding white noise to test patterns corresponding to atomic propositions. With the learned if–then rules described in Section 3.1.1, we examine whether the system can reason correctly from the noisy data. To generate the noisy data, we first take one test pattern T from the test dataset in Section 3.1.2. Then we generate a random vector W satisfying formula (10) and formula (11); at last, we add the random vector W to the test pattern T and obtain a noisy testing pattern $T' = T + W$. In all, 200 noisy patterns are generated for each fact in each noise level, which means that the system obtains 3400 noisy patterns of the same noise level. Here, the noise level is defined according to the signal-to-noise ratio (SNR)

$$SNR = 10\log_{10}\frac{P_{signal}}{P_{noise}}, \tag{12}$$

where P is the average power.

Fig. 7 presents the reasoning result. The figure shows that a boundary SNR exists and that the system can produce correct reasoning results when the SNR of a fact is greater than the boundary. For example, when $\delta_r = 0.3$ and $SNR > 17$, the correct reasoning ratio is 100% for the noisy data. Results of this experiment demonstrate that the proposed method is robust to noisy data.

From Section 3.1.2, we know that, for test patterns that differ from the training patterns (but which obey the same distribution), the proposed method can yield good reasoning results. This section also describes that, using noisy test patterns that differ greatly from the training data, the reasoning results are good. Consequently, the proposed method has good generalization ability.

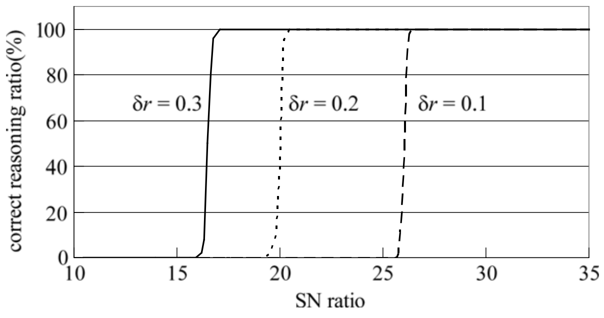

Fig. 7. Ratio of correct reasoning when noisy data are given.

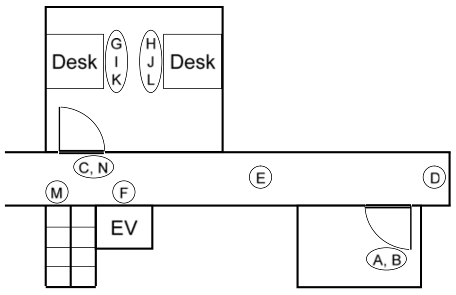

Fig. 8. The building layout, showing where the photographs of Fig. 9 were taken.

3.2. Simulation using real-world data

In this experiment, the proposed system learns the sequentially provided if–then rules; the atomic propositions of such rules are photographs taken from a real environment. We use photographs from one floor of a building to test the proposed method. Fig. 8 depicts the floor layout; Fig. 9 presents some photographs taken from that floor. The alphabet letters enclosed by circles in Fig. 8 represent places where the photographs in Fig. 9 were taken.

In all, 14 objects are portrayed in Fig. 9. For every object, we took 20 photographs from different angles, yielding 280 images for 14 objects. Each image is 56 × 46 pixels.

The if–then rules for the systems to learn are "A→B", "B→ D∨E", "E→ (C∧N)∨F∨M", "(C∧N) → (G∧I)∨(H∧J)", "(G∧I) → K", and "(H∧J) → L". They mean "the closed door → the open door", "the open door → the wall ∨ the hallway", "the hallway → (the laboratory ∧ the room plate) ∨ the elevator ∨ the down stairs", "the laboratory ∧ the room plate→(the desk 1 ∧ the closed drawer 1)∨(the desk 2 ∧ the closed drawer 2)", "(the desk 1 ∧ the closed drawer 1)→ the open drawer 1", and "(the desk 2 ∧ the closed drawer 2)→ the open drawer 2".

The original six if–then rules are resolved to 10 basic if–then rules: A→B, B→D, B→E, E→(C∧N), E→F, E→M, (C∧N)→(G∧I), (C∧N)→(H∧J), (G∧I)→K, and (H∧J)→L. For every basic if–then rule, 400 patterns are generated and each atomic proposition of which is selected randomly from 20 different-angled photos. For example, for one if–then rule A→B, we randomly choose 1 image from 20 photographs of area A as the conditional part, and randomly choose 1 image from 20 photographs of area B as the sequential part. In all, there are 4000 patterns for all 10 basic if–then rules. Such patterns are input sequentially to the system under the incremental mode, i.e. during learning, patterns representing new rules are input to the system incrementally. The parameters are set to $\Lambda_{edge} = 100$, $\lambda = 50$ for the learning process.

Table 4 presents learning results. The input 4000 if–then rules are categorized into 10 different clusters (Column II of Table 4) in long-term memory; each cluster has members corresponding respectively only to A→B, B→D, B→E, E→(C∧N), E→F, E→M, (C∧N)→(G∧I), (C∧N)→(H∧J), (G∧I)→K, and (H∧J)→L. The

(A). close door (B). open door (C). laboratory

(D). wall (E). hallway (F). elevator

(G). desk-1 (H). desk-2 (I). desk drawer-1 (close)

(J). desk drawer-2 (close) (K). desk drawer-1 (open) (L). desk drawer-2 (open)

(M). down stairs (N). room plate

Fig. 9. Some images for the experiment using real-world data.

Table 4
Learning results for real world data.

Cluster no.	If–then rule	Number of members	Compression rate (%)
1	$A \to B$	81	20.25
2	$B \to D$	135	33.75
3	$B \to E$	46	11.5
4	$E \to (C \wedge N)$	59	14.75
5	$E \to F$	28	7.0
6	$E \to M$	136	34.0
7	$(C \wedge N) \to (G \wedge I)$	78	19.5
8	$(C \wedge N) \to (H \wedge J)$	104	26.0
9	$(G \wedge I) \to K$	131	32.75
10	$(H \wedge J) \to L$	114	28.5
Total	–	912	22.8

labels of such clusters are the same rules as those used in a symbol-based manner, meaning that the system is able to categorize if–then rules appropriately. The quantities of members of every cluster are presented in Column III of Table 4. The total members representing the 10 basic if–then rules are 912. Column IV of Table 4 presents the compression ratio, although the learning data included 4000 if–then rules with mutually differing atomic propositions, only 912 rules are stored in long-term memory. The system eliminated 3088 rules as similar rules to previously stored rules: the total compression ratio is 22.8%. The proposed method avoids memory consumption through elimination of similar data.

The system obtains learning data sequentially and incrementally. Therefore, the result of learning indicates that the system learns new data without collapsing previously learned data: the proposed method can realize incremental learning well for real world data.

Regarding the reasoning phase, the parameter is set to $\delta_r = 0.3$. We first present one photograph taken from area A of Fig. 8 (which is a newly obtained photograph, different from the training

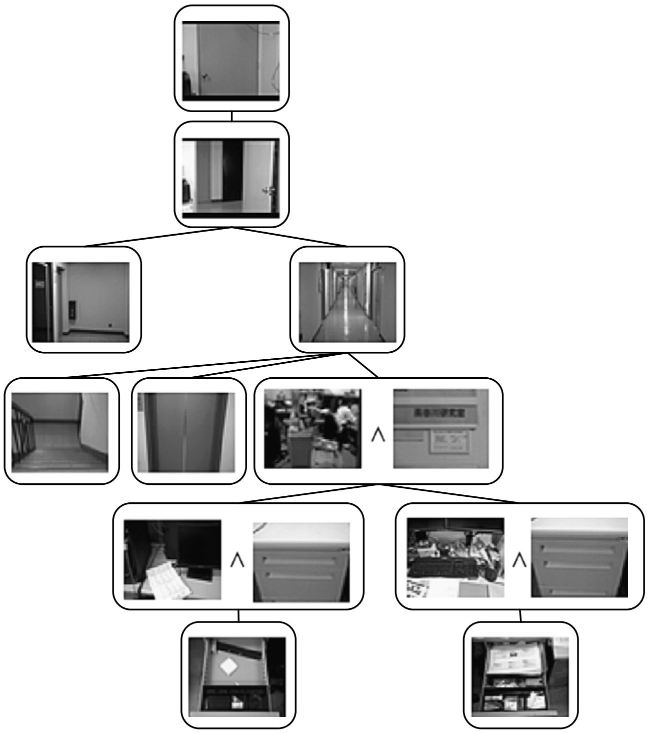

Fig. 10. Reasoning results for real world data.

images) as a fact to test whether the system can produce inferences correctly. With the input "closed door" image, the system outputs a tree, as portrayed in Fig. 10, as the reasoning result. From this tree, we know that by reasoning from the fact of "closed door", the system knows the path from "closed door" to "open door", then "down stairs", "elevator", etc. Comparing the generated tree after reasoning with the learned if–then rules, there are neither omissions nor duplication of if–then rules in this tree. The salient implication is that the proposed method can produce inferences well. This result shows that the proposed system is effective for intelligence systems to solve various tasks autonomously in a real environment.

From the reasoning results obtained from "closed drawers", we can assess the importance of storing an if–then rule with conjunction. If the system is impossible to address the conjunction, it would be unable to produce a correct inference because the closed drawers appear to be almost identical.

Through classification of the if–then rules to clusters, the proposed method avoids duplication of reasoning results. If we do no clustering for if–then rules (i.e. if we force the system to regard every if–then rule in long-term memory as belonging to different clusters), then, in all, 500,376 nodes are created in the output tree.

In this experiment, we also provide A, B, D, E, $C \wedge N$, F, M, $G \wedge I$, $H \wedge J$, K, and L as facts to test whether the system can produce inferences correctly. For each fact, 20 patterns are generated by taking photographs from the corresponding area of Fig. 8 (new photographs different from the photographs used for training). In all, the system has 220 patterns. The reasoning phase will output an OR-tree when the pattern of a fact is presented. We judge whether this fact is reasoned correctly or not by comparing the output OR-tree with Fig. 10. The correct reasoning ratio for all test patterns is 99.9%, which also testifies to the good generalization capability of the proposed method.

4. Conclusion

We proposed a reasoning system that can learn pattern-based if–then rules including conjunction, disjunction, and negation.